ABSTRACT

This dissertation investigates the dual application of deep learning in the domains of handwritten digit recognition & artistic style transfer, highlighting the potential of neural networks in both practical & creative fields. The project is structured around two core components: a Convolutional Neural Network (CNN) for digit recognition & a style transfer technique leveraging the VGG19 network.

The first component centers on developing a CNN model for the recognition of handwritten digits from the MNIST dataset. The model architecture includes multiple convolutional layers followed by max-pooling layers, capturing essential features of the input images. The final classification is performed by dense layers, with the model achieving high accuracy in distinguishing between the ten digit classes. This system not only demonstrates the efficacy of CNNs in image classification tasks but also lays the groundwork for understanding more complex neural network architectures.

The second component explores artistic style transfer, an innovative application of neural networks that merges the content of an image with the style of an artwork. Utilizing the VGG19 network, pre-trained on a large dataset, the style transfer process extracts style features from the artwork and content features from the target image. By optimizing a combined loss function, the model generates new images that blend the content and style seamlessly. This technique provides artists with a powerful tool to visualize and experiment with various art styles, promoting creative exploration.

The digit recognition system offers practical insights into image classification, while the style transfer system opens new avenues for artistic expression. This covers the detailed implementation, training procedures, & performance evaluation of both models, discussing their implications & potential improvements. The project exemplifies the fusion of technical prowess & artistic creativity, demonstrating the transformative impact of neural networks.

TABLE OF CONTENTS

OBJECTIVE

1.1: General Description of the Project

'Network Intrusion Detector' is built using machine learning. This system will analyze the network security of the system. It contains datasets and algorithms. The attacks are classified into four categories and then they are trained and tested by NSL-KDD training and testing datasets and are further processed using machine learning algorithms and ensemble learning concept.

1.2: User Requirements

1.2.1: User Requirements

Hardware Requirements:

- Memory: 8GB (minimum) 4GB (required)
- GPU: 6GB (minimum)
- Platform: Windows 8 or later
- Processor: Intel Pentium i5 or later
- Internet Connection: Not Required

Software Requirements

- Platform: Python (v-3.10.0)
- Web Tool: Jupyter Notebook
- Web Browser: Chrome, Mozilla, etc

1.2.2: Problems Faced by the Existing Work

- High memory usage
- No GUI
- Time consuming

Review of Literature

This literature review aims to provide a comprehensive overview of the research landscape surrounding neural style transfer and digit recognition, exploring the evolution of these technologies, the integration of advanced methodologies such as convolutional neural networks (CNNs), of research and development in these fields.

Neural Style Transfer

Neural style transfer is a technique that combines the content of one image with the style of another using deep learning algorithms. The pioneering work by **Gatys, Ecker, and Bethge (2016)** introduced the concept of neural style transfer using CNNs, specifically the VGG-19 network. This method involves optimizing an image to match the content representation from one image and the style representation from another. The effectiveness of this approach in generating aesthetically pleasing images has been widely acknowledged in subsequent research (Gatys et al., 2016).

Digit Recognition

Digit recognition, particularly using the MNIST dataset, is a well-established task in the field of machine learning. The MNIST dataset, introduced by **LeCun et al. (1998)**, comprises handwritten digits from 0 to 9 and has become a benchmark for evaluating the performance of various classification algorithms.

Integration with User Interfaces & Educational Applications

- The integration of NST with user-friendly interfaces has been a significant development. The use of libraries such as **Tkinter** and **Gradio** to create accessible GUIs allows users with minimal technical expertise to apply artistic styles to images. This has made neural style transfer tools widely available to artists, designers, and educators, enhancing creative expression.
- The development of interactive UIs for digit recognition has expanded its application in educational contexts. Tools that allow children to draw digits on a canvas and receive real-time feedback provide an engaging way for young learners to practice and learn numbers. Additional features, such as displaying fun facts about the recognized digit, to make learning more enjoyable.

2.1: Methodology

The methodology for developing the Neuro-Style Transfer and Digit Recognition projects involves several key steps, including data preparation, model implementation, and user interface development. This section outlines the detailed process for each component.

Neuro-Style Transfer

Importing Libraries

- Essential Python libraries are imported, including numpy for numerical operations, tensorflow for building and training neural networks, and tkinter for creating the graphical user interface (GUI).

Model and Data Preparation

- The VGG-19 model pre-trained on the ImageNet dataset is used for neural style transfer. This model is particularly effective in capturing high-level features necessary for style and content extraction.

Style Transfer Algorithm

- The content and style representations are extracted from the intermediate layers of the VGG-19 model. Specifically, deeper layers capture the content while shallower layers capture the style.
- A loss function combining content loss and style loss is defined. Content loss ensures the output image retains the structure of the content image, while style loss ensures the output image mimics the texture and color patterns of the style image.

User Interface Development

- A user-friendly interface is created using the Tkinter library. This interface allows users to upload content and style images, initiate the style transfer process, and visualize the resulting image.
- The UI includes options for users to adjust parameters such as the number of iterations and the weights assigned to content and style losses, providing flexibility and control over the style transfer process.

Digit Recognition

Importing Libraries

- Essential Python libraries are imported, including numpy for numerical operations, tensorflow for building the convolutional neural network (CNN), and tkinter for creating the GUI.

Dataset Preparation

- The MNIST dataset, which contains 60,000 training images and 10,000 testing images of handwritten digits, is used. The dataset is loaded and preprocessed to normalize pixel values to a range of 0 to 1, enhancing the performance of the neural network.

Model Implementation

- A CNN is constructed for digit recognition. The architecture includes multiple convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification.
- The model is compiled with the Adam optimizer and categorical cross-entropy loss function. The training process involves iterating over the dataset to minimize the loss and improve accuracy.

User Interface Development

- An interactive canvas is created using Tkinter, allowing users to draw digits directly. The drawn digits are captured and fed into the trained CNN for recognition.
- The UI is designed to be intuitive, with features that display the recognized digit and provide educational content such as fun facts about the digit. This makes the tool engaging for young children.

Features

The Neuro-Style Transfer and Digit Recognition projects are designed to provide powerful yet user-friendly tools for creative and educational purposes. Below are the key features of each project:

Neuro-Style Transfer

- Necessary but useful Neural Style TransferUser-Friendly Interface
- Developed with the Tkinter library, the graphical user interface (GUI) allows users to easily upload content and style images.Real-Time Visualization
- Displays the generated image in real time, allowing users to see the effects of different styles immediately.
- The interface includes tools to save and export the generated images for further use.
- Designed to be used by a wide range of users, from artists and designers to educators and students.
- The intuitive UI ensures that even users with minimal technical knowledge can perform neural style transfers.
- Customizable Parameters

Digit Recognition

- Utilizes a convolutional neural network (CNN) trained on the MNIST dataset to accurately recognize handwritten digits from 0 to 9.
- Interactive Drawing Canvas
- Educational Tool designed for children aged 2.5 and above, the application provides an engaging way to learn digits.
- Displays fun facts and educational content about the recognized digits, making learning enjoyable and informative.
- Ease of Use
- The application can be used in educational settings to teach digits in an interactive manner.

Benefits

General Benefits

- Open Source and Community Support
- Future Extensibility
- Accessibility Through Gradio

Neuro-Style Transfer

- Enhanced Creativity
- User-Friendly Experience
- Versatility
- Educational Value
- Accessibility
- Customizable

Digit Recognition

- Interactive Learning
- Educational Enhancement
- Ease of Use
- High Accuracy
- Engagement and Motivation
- Parental and Teacher Aid

Modules

The Neuro-Style Transfer and Digit Recognition projects are composed of several key modules, each responsible for different aspects of the overall functionality. Below is an overview of the main modules for both projects:

Neuro-Style Transfer

- Image Preprocessing Module
- Feature Extraction Module
- Style Transfer Algorithm Module
- User Interface Module
- Visualization Module
- Web Interface Module

Digit Recognition

- Preprocessing Module
- Model Building Module
- Model Training Module
- Drawing Canvas Module
- Recognition & Facts Module
- Web Interface Module

Hardware Requirements

Hardware Requirements:

- Memory: 8GB (minimum) 4GB (required)
- GPU: 6GB (minimum), CUDA support
- Platform: Windows 8 or later
- Processor: Intel Pentium i5 or later
- Internet Connection: Not Required

Software Requirements

Software Requirements

- Platform: Python (v-3.10.0)
- Web Browser: Chrome, Mozilla, etc
- Additional tools: Anaconda, Visual Studio
- OS: Windows, Linux
- Libraries: tensorflow(2.10) and keras compatible to cuda

Intended Users

Neural Style Transfer

- Artists and Designers
- Digital Content Creators
- Educators and Students in Art and Design
- Machine Learning Enthusiasts and Researchers

Digit Recognition

- Young Children and Early Learners(Under Parental Guidance)
- Parents and Teachers
- Educational Institutions

Tools Used

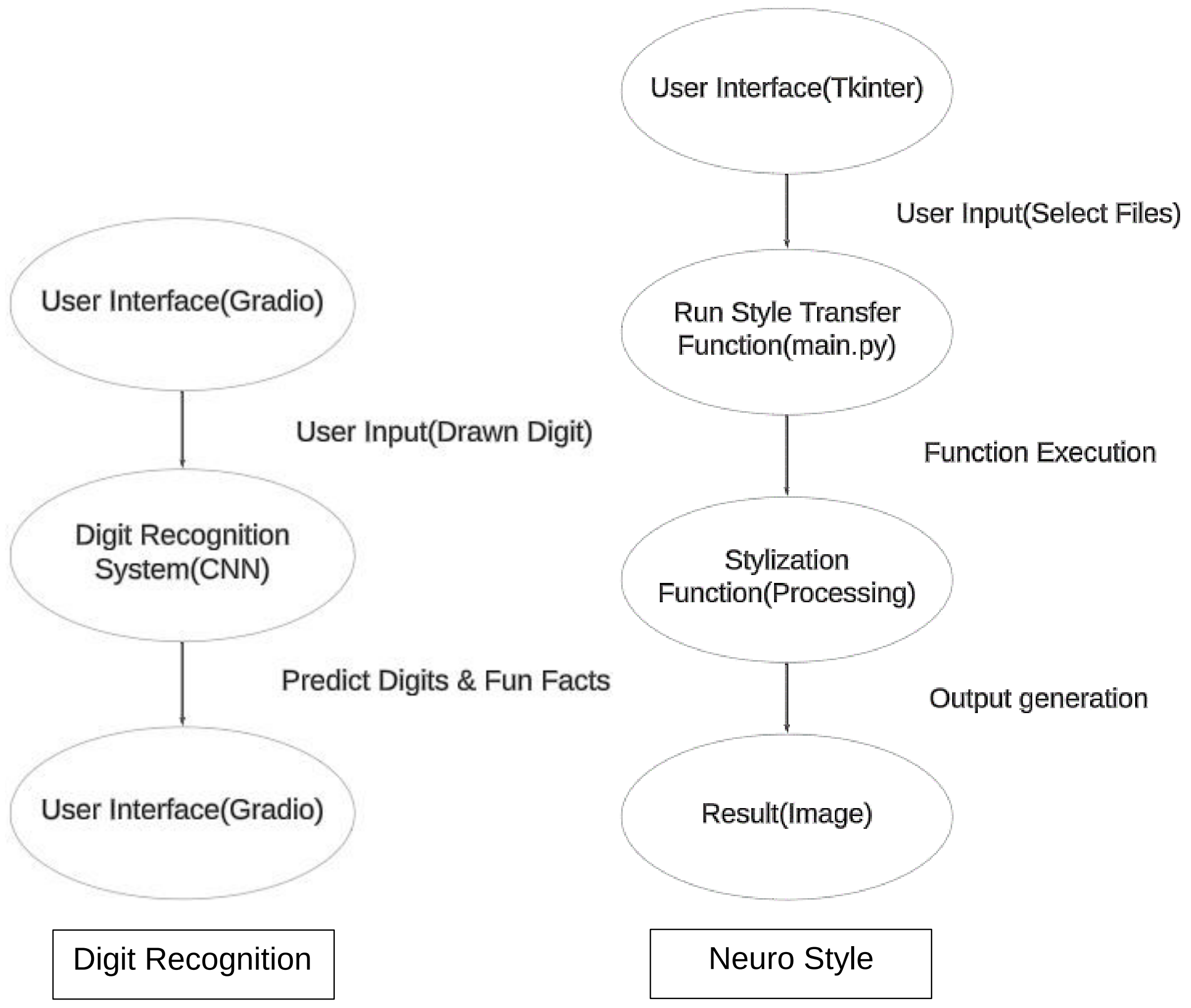
- Programming Tools: Python, ML

0 – Level DFD

The Level 0 DFD provides a high-level overview of the system, showing the system as a single process and its interactions with external entities.

External Entities:

- User: The person interacting with the system, providing input images and receiving output.
- Processes:
- Neuro-Style Transfer and Digit Recognition System: The main process of the system.
- Data Flows:
- Input Image: Image provided by the user for style transfer or digit recognition.
- Stylized Image/Recognized Digit: The output image after style transfer or the recognized digit returned to the user.



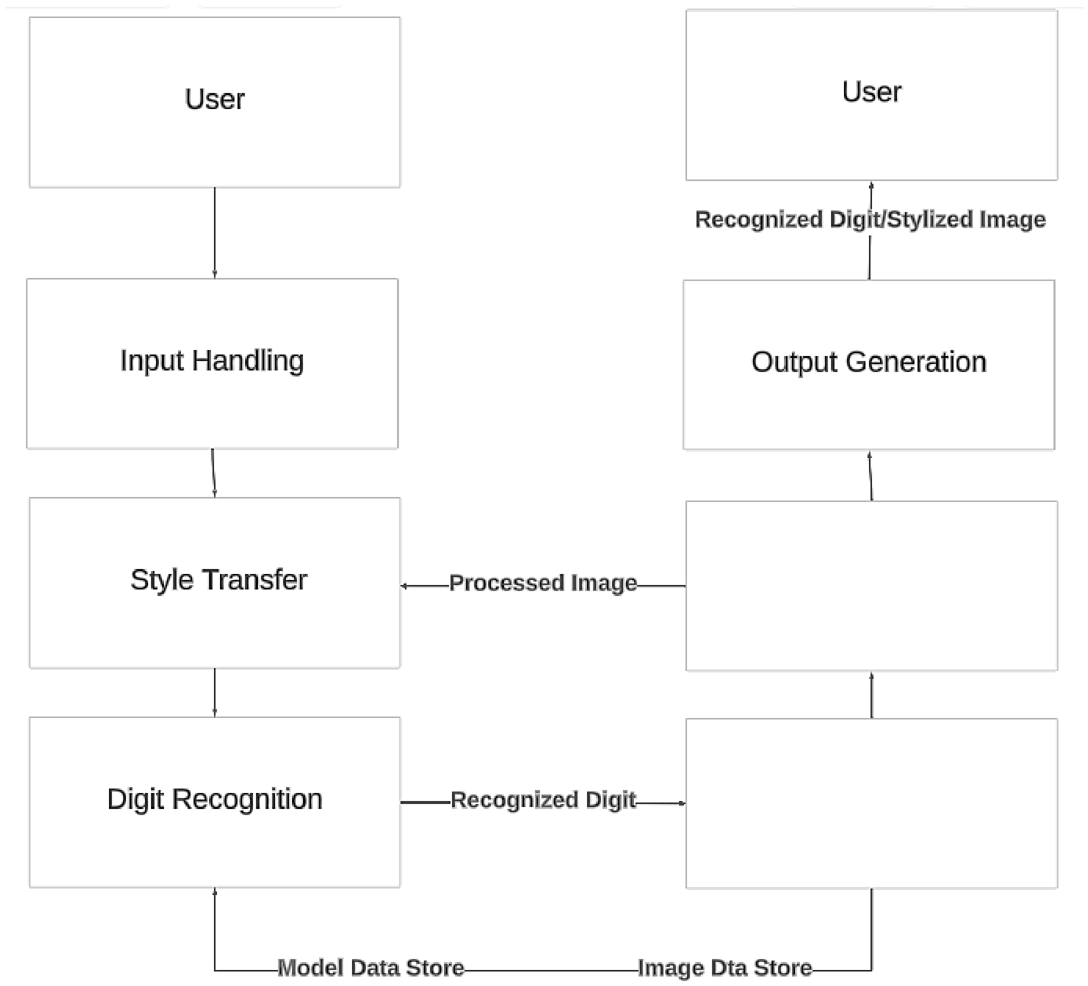
1 – Level DFD

Processes:

- Input Handling: Process to handle user input images.
- Style Transfer: Process to apply the neural style transfer.
- Digit Recognition: Process to recognize digits from user-drawn input.
- Output Generation: Process to provide the stylized image or recognized digit to the user.

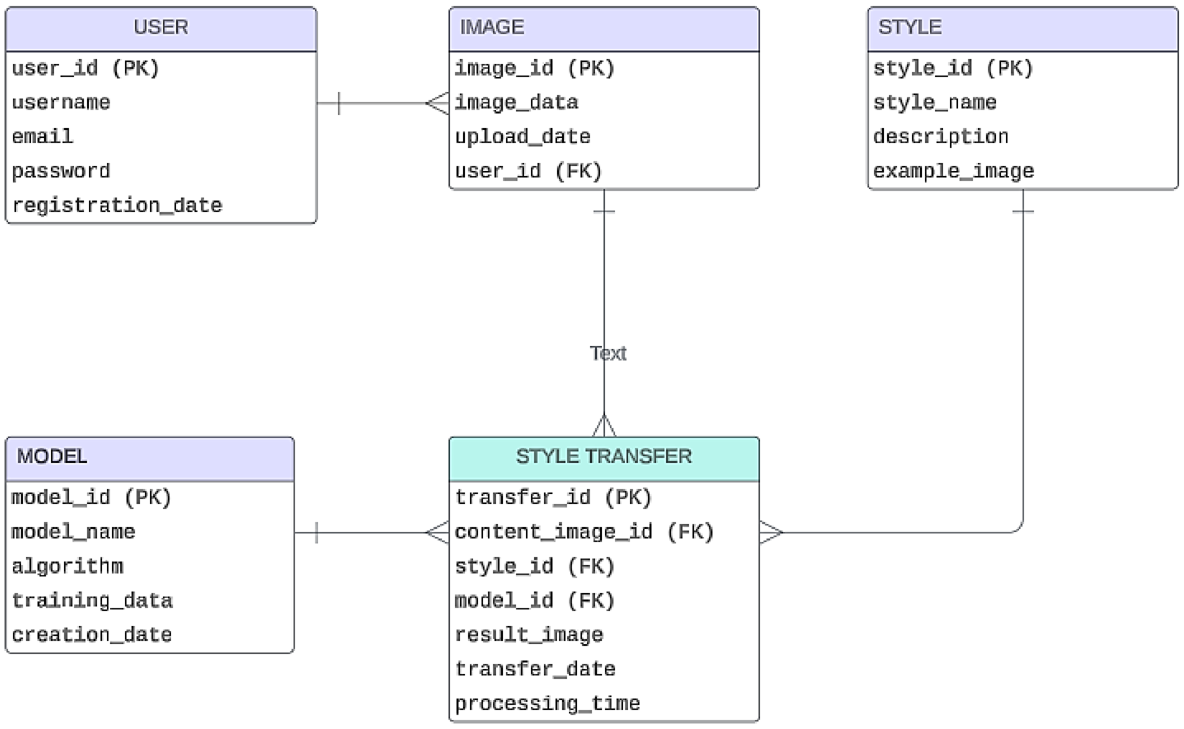
Data Flows:

- Input Image: Image provided by the user.
- Processed Image: Intermediate image after style transfer.
- Recognized Digit Data: Result of digit recognition.
- Stylized Image / Recognized Digit: Final output to the user.

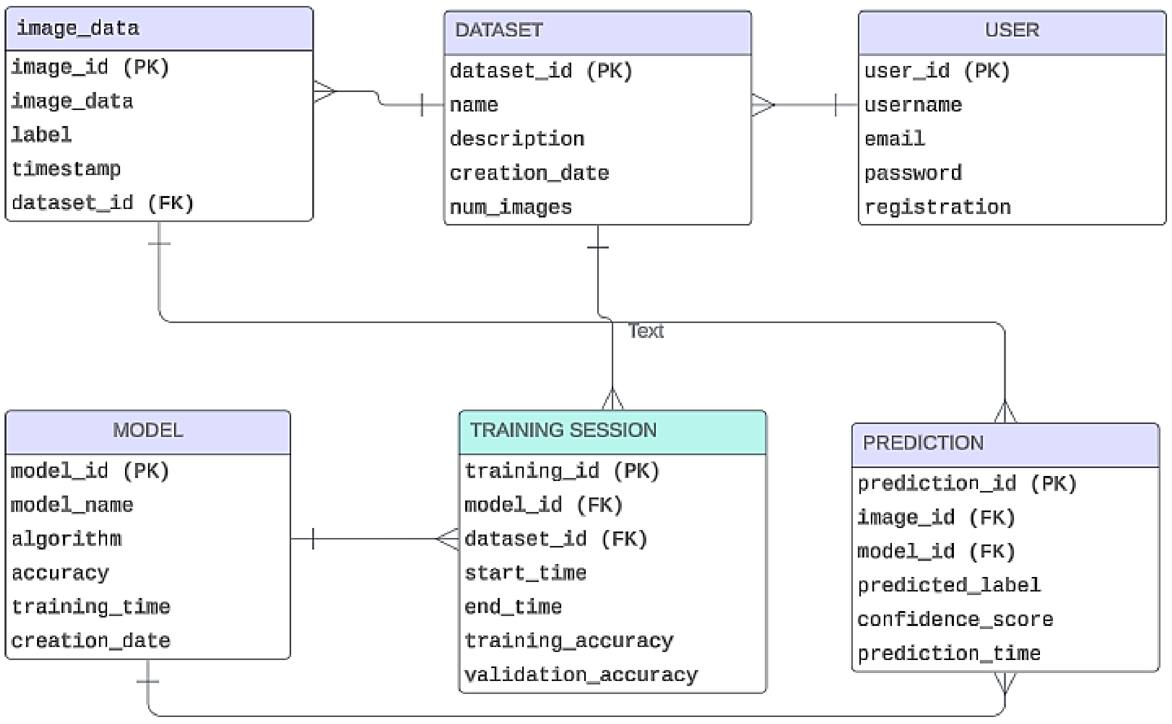


ER-Diagram

Neuro Style



Digit Recognition



Process Involved

Identify the Processes

Import Libraries

Libraries in Python are collections of pre-written code modules or functions that extend the capabilities of the Python programming language. These libraries provide ready-to-use tools and functionalities for various tasks. Some of those are:

NumPy

- NumPy, short for Numerical Python, is a fundamental library for numerical computing in Python. It provides support for arrays, matrices, and a large collection of mathematical functions to operate on these data structures. NumPy is highly optimized for performance and is widely used in scientific computing, data analysis, and machine learning. In the context of this project, NumPy is used for handling image data and performing mathematical operations necessary for neural network computations.

TensorFlow

- TensorFlow is an open-source machine learning library developed by Google. It provides a comprehensive ecosystem for building, training, and deploying machine learning models. TensorFlow supports a wide range of neural network architectures and includes tools for both high-level and low-level programming. For this project, TensorFlow is utilized to implement and train the convolutional neural networks (CNNs) used in both the Neuro-Style Transfer and Digit Recognition tasks.

PIL (Pillow)

- PIL, the Python Imaging Library, has been succeeded by Pillow, which is a more actively maintained and user-friendly version. Pillow adds image processing capabilities to Python, allowing for tasks such as opening, manipulating, and saving various image file formats. It supports a wide range of image operations, including resizing, cropping, filtering, and image transformations. In this project, Pillow is used to preprocess images before they are fed into the neural networks, as well as to display the results of the style transfer and digit recognition processes.

Tkinter:

- Tkinter is the standard Python interface to the Tk GUI toolkit. It is included with Python and provides a straightforward way to create graphical user interfaces. Tkinter allows developers to build windows, dialogs, buttons, and other GUI elements with ease. In this project, Tkinter is used to create user-friendly interfaces for both the Neuro-Style Transfer and Digit Recognition applications. These interfaces enable users to interact with the underlying machine learning models without needing to understand the technical details.

Gradio

- Gradio is a library that allows developers to create customizable user interfaces for machine learning models quickly and easily. It provides a web-based interface for interacting with models, enabling users to upload images, input text, and see results in real time. Gradio supports various input and output types and can be easily integrated into existing Python code. In this project, Gradio is used to create an interactive web-based interface for both the Neuro-Style Transfer and Digit Recognition applications. This interface allows users to interact with the models through their web browsers, making the applications more accessible and easier to use without requiring any installation or setup.

Keras:

- Keras is a high-level neural network API written in Python. It provides a user-friendly interface for building, training, and deploying neural network models. With Keras, you can quickly prototype deep learning models, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), & more, facilitating the implementation of machine learning algorithms in your project. Keras seamlessly integrates with TensorFlow, making it a powerful tool for developing advanced neural network architectures & optimizing model performance. Through Keras, leveraged the capabilities of deep learning to enhance the functionality of project, enabling tasks such as style transfer & digit recognition.

Import Dataset

Neuro-Style Transfer

Model and Data Preparation

- The VGG-19 model pre-trained on the ImageNet dataset is used for neural style transfer. This model is particularly effective in capturing high-level features necessary for style and content extraction.

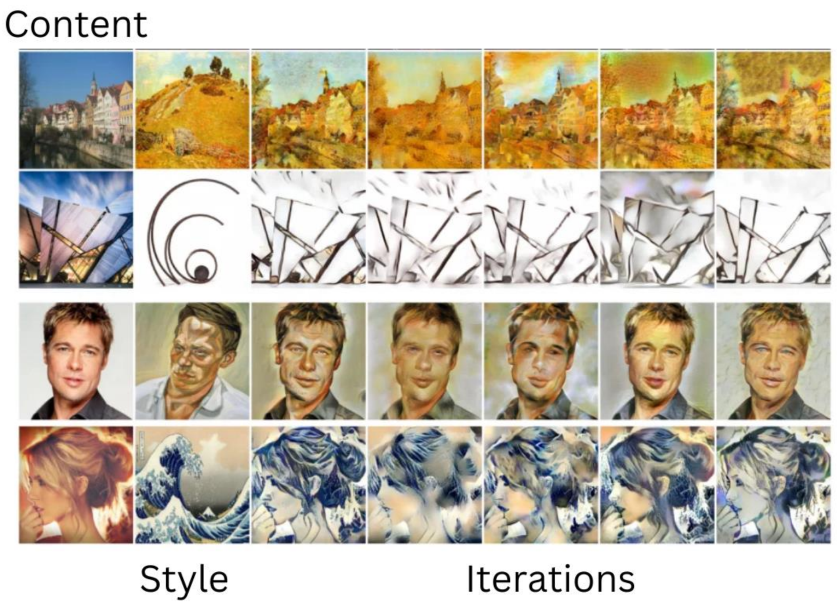
Digit Recognition

Model Implementation & Dataset Preparation

- The MNIST dataset, which contains 60,000 training images and 10,000 testing images of handwritten digits, is used. The dataset is loaded & preprocessed to normalize pixel values to a range of 0 to 1, enhancing the performance of the neural network.
- A CNN is constructed for digit recognition. Architecture includes multiple convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification.
- The model is compiled with the Adam optimizer and categorical cross-entropy loss function. The training process involves iterating over the dataset to minimize the loss and improve accuracy.

Testing

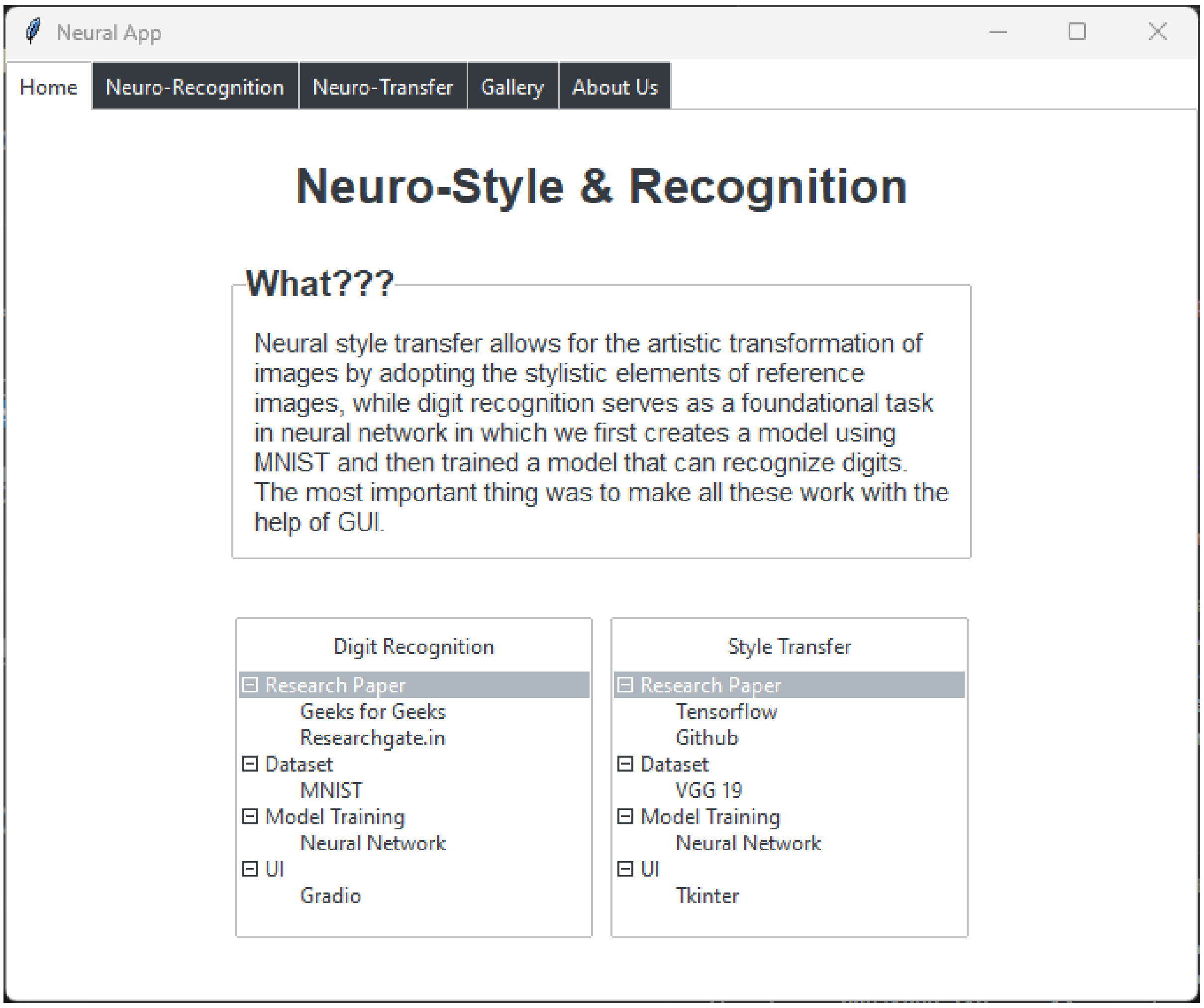
Neural Style Transfer



Digit Recognition

Text	**Output**	**Actual Output(April)**	**Actual Output(May)**	**Result**
	0	-	0	Successful
	1	-	1	Successful
	2	7	2	Successful
	3	8	3	Successful
	4	9	4	Successful
	5	3	5	Successful
	6	9	6	Successful
	7	3	7	Successful
	8	3	8	Successful
	9	9	9	Successful

Admin



Neuro-Style & Recognition

What???

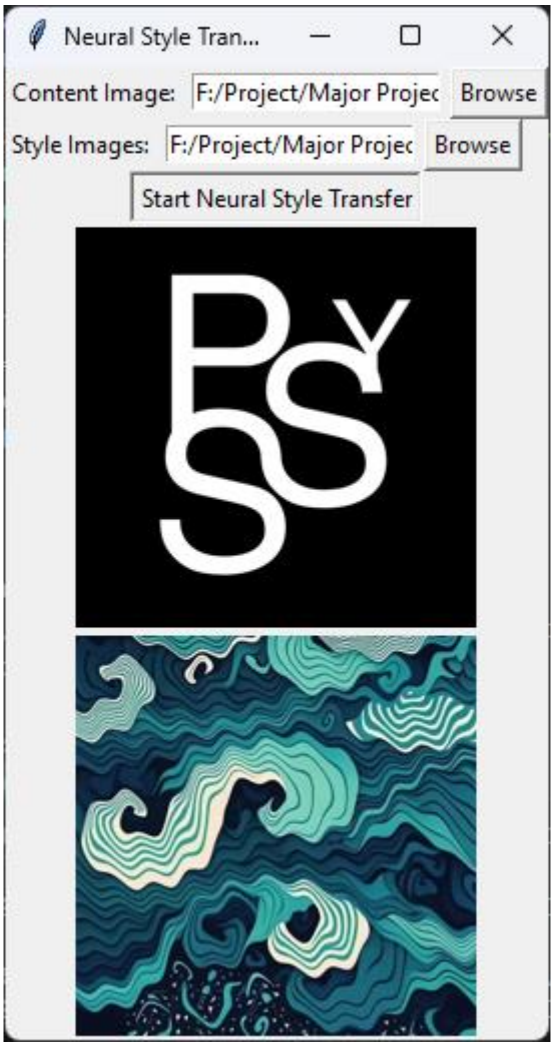
Neural style transfer allows for the artistic transformation of images by adopting the stylistic elements of reference images, while digit recognition serves as a foundational task in neural network in which we first creates a model using MNIST and then trained a model that can recognize digits. The most important thing was to make all these work with the help of GUI.

Digit Recognition
- Research Paper
  - Geeks for Geeks
  - Researchgate.in
- Dataset
  - MNIST
- Model Training
  - Neural Network
- UI
  - Gradio

Style Transfer
- Research Paper
  - Tensorflow
  - Github
- Dataset
  - VGG 19
- Model Training
  - Neural Network
- UI
  - Tkinter

```
import tkinter as tk
import ttkbootstrap as ttk
import subprocess
import threading
import os
from ttkbootstrap.constants import *
from PIL import Image, ImageTk

#class for app window
class NeuralApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Neural App")

        # Calculate window dimensions
        screen_width = master.winfo_screenwidth()
        screen_height = master.winfo_screenheight()
        window_width = int(screen_width * 0.5) #680
        window_height = int(screen_height * 0.7) #537
        # print(window_height, window_width)

        # Set window size
        self.master.geometry(f"{window_width}x{window_height}")

        # Create a Notebook (multi-tabbed window)
        self.notebook = ttk.Notebook(self.master, bootstyle="dark")
        self.notebook.pack(fill=tk.BOTH, expand=True)

        # Create and add tabs to the Notebook
        self.home_tab = ttk.Frame(self.notebook)
        self.neuro_rec_tab = ttk.Frame(self.notebook)
        self.neuro_tran_tab = ttk.Frame(self.notebook)
        self.gallery_tab = ttk.Frame(self.notebook)
        self.about_us_tab = ttk.Frame(self.notebook)


        self.notebook.add(self.home_tab, text="Home")
        self.notebook.add(self.neuro_rec_tab, text="Neuro-Recognition")
        self.notebook.add(self.neuro_tran_tab,text="Neuro-Transfer")
        self.notebook.add(self.gallery_tab, text="Gallery")
        self.notebook.add(self.about_us_tab, text="About Us")

#Creating widgets for various tabs
        self.create_home_widgets() #Create widgets for the Home tab
        self.create_neuro_rec_widgets() #Create widgets for Recognition tab
        self.create_neuro_tran_widgets() # Create widgets for the NST tab
        self.create_gallery_widgets() # Create widgets for the Gallery tab
        self.create_about_us_widgets() # Create widgets for the About Us tab

#Creating Home Page
    def create_home_widgets(self):
        # Heading
        heading_label    =    ttk.Label(self.home_tab,    text="Neuro-Style    &
Recognition", font=("Helvetica", 20, "bold"))
        heading_label.pack(pady=25)

        # Label frame with text
        label_frame = ttk.LabelFrame(self.home_tab, text="What???")
        label_frame.pack(pady=0)
        # Configure font and size for the label frame text
        s = ttk.Style()
        s.configure('TLabelframe.Label', font=('Helvetica', 15, 'bold'))

        info_label = ttk.Label(label_frame, text="Neural style transfer allows
for the artistic transformation of images by adopting the stylistic elements of
reference images, while digit recognition serves as a foundational task in
neural network in which we first creates a model using MNIST and then trained
a model that can recognize digits. The most important thing was to make all
these work with the help of GUI.", wraplength=400)
        info_label.pack(padx=10, pady=10)
        # Configure font and size for the content text
        s = ttk.Style()
        s.configure('TLabel', font=('Helvetica', 7))

        # Buttons aligned in a line
        button_frame = ttk.Frame(self.home_tab)
        button_frame.pack()
        # Digit Recognition Tree View
        self.digit_recognition_tree = ttk.Treeview(button_frame)
        self.digit_recognition_tree.heading("#0", text="Digit Recognition")  # 
Set heading

        # Inserting top-level items
        self.digit_recognition_tree.insert("", "end", text="Research Paper")
        self.digit_recognition_tree.insert("", "end", text="Dataset")
        self.digit_recognition_tree.insert("", "end", text="Model Training")
        self.digit_recognition_tree.insert("", "end", text="UI")

        # Get the item ID of the "Dataset" item
        research_id = self.digit_recognition_tree.get_children()[0]
        dataset_id = self.digit_recognition_tree.get_children()[1]
        model_id = self.digit_recognition_tree.get_children()[2]
        ui_id = self.digit_recognition_tree.get_children()[3]

        # Insert "MNIST" under "Dataset"
        self.digit_recognition_tree.insert(research_id, "end", text="Geeks for 
Geeks")
        self.digit_recognition_tree.insert(research_id,                 "end", 
text="Researchgate.in")
        self.digit_recognition_tree.insert(dataset_id, "end", text="MNIST")
        self.digit_recognition_tree.insert(model_id,    "end",    text="Neural 
Network")
        self.digit_recognition_tree.insert(ui_id, "end", text="Gradio")

        # Packing the TreeView
        self.digit_recognition_tree.pack(side=tk.LEFT, padx=5, pady=10)

        # Neural Style Transfer Button
        self.digit_recognition_tree = ttk.Treeview(button_frame)
        self.digit_recognition_tree.heading("#0", text="Style Transfer")  # Set 
heading

        # Inserting top-level items
        self.digit_recognition_tree.insert("", "end", text="Research Paper")
        self.digit_recognition_tree.insert("", "end", text="Dataset")
        self.digit_recognition_tree.insert("", "end", text="Model Training")
        self.digit_recognition_tree.insert("", "end", text="UI")

        # Get the item ID of the "Dataset" item
        research_id = self.digit_recognition_tree.get_children()[0]
        dataset_id = self.digit_recognition_tree.get_children()[1]
        model_id = self.digit_recognition_tree.get_children()[2]
        ui_id = self.digit_recognition_tree.get_children()[3]

        # Insert "MNIST" under "Dataset"
        self.digit_recognition_tree.insert(research_id,                 "end", 
text="Tensorflow")
        self.digit_recognition_tree.insert(research_id, "end", text="Github")
        self.digit_recognition_tree.insert(dataset_id, "end", text="VGG 19")
        self.digit_recognition_tree.insert(model_id,    "end",    text="Neural 
Network")
        self.digit_recognition_tree.insert(ui_id, "end", text="Tkinter")
        # Packing the TreeView
        self.digit_recognition_tree.pack(side=tk.LEFT, padx=5, pady=10)

        # Centering button_frame within the Home tab
        button_frame.pack(expand=True)
        button_frame.place(relx=0.5, rely=0.75, anchor=tk.CENTER)

    #function to access files that need to be executed when clicked
    def run_digit_recognition(self):
        def run_command_dr():
            # Run the digit recognition script
            subprocess.run(["python",  "./Major  Project  -  Neuro  Style  and
Recognition/digit/ui_gradio.py"])
            # Close the tkinter window
            self.master.destroy()

        # Start a new thread to run the command
        command_thread = threading.Thread(target=run_command_dr)
        command_thread.start()
    def run_neural_style_transfer(self):
    # Define the function to run the command
        def run_command_nst():
            subprocess.run(["python",  "./Major  Project  -  Neuro  Style  and
Recognition/nst/gui_tkinter.py"])
            # Close the tkinter window
            self.master.destroy()

        # Start a new thread to run the command
        command_thread = threading.Thread(target=run_command_nst)
        command_thread.start()
#Home page Over

#Creating Neuro-Recognition Page
    def create_neuro_rec_widgets(self):
        # Heading
        heading_label   =   ttk.Label(self.neuro_rec_tab,   text="Neuro-Digit
Recognition", font=("Helvetica", 20, "bold"))
        heading_label.pack(pady=25)

        # Load the image
        image_path    =    "./Major    Project    -    Neuro    Style    and
Recognition/icons/mnist.jpeg"  # Replace with the actual path to your image
        image = Image.open(image_path)
        image = image.resize((275, 187))  # Adjust the size as necessary
        tk_image = ImageTk.PhotoImage(image)

        # Create a label for the image
        image_label = ttk.Label(self.neuro_rec_tab, image=tk_image)
        image_label.image  =  tk_image   #  Keep  a  reference  to  avoid  garbage
collection
        image_label.pack(pady=0)

        # Label frame with text
        label_frame    =    ttk.LabelFrame(self.neuro_rec_tab,    text="Digit
Recognition")
        label_frame.pack(pady=10)
        info_label = ttk.Label(label_frame, text="Users can draw digits on a
canvas provided within the application. Once the digit is drawn, the trained
model embedded within the application will analyze the input and provide
recognition of the digit drawn.\n   >> Model Train\n   >> Canvas Drawing \n   >>
Model Recognition \n   >> Recognition Result", wraplength=400)
        info_label.pack(padx=10, pady=10)

        # Configure font and size for the label frame text
        s = ttk.Style()
        s.configure('TLabelframe.Label', font=('Helvetica', 15, 'bold'))
        # Configure font and size for the content text
        s.configure('TLabel', font=('Helvetica', 11))

        button    =    ttk.Button(self.neuro_rec_tab,    text="Let's    Try!!",
command=self.run_digit_recognition)
        button.pack(side=tk.RIGHT, padx=(5, 10), pady=0)  # Place on the right
side
        button.place(relx=0.793, rely=0.88, anchor=tk.NE)  # Place on the corner
of the label frame
#Digit Recognition Page Over

#Creating Neuro-Style-Transfer Page
    def create_neuro_tran_widgets(self):
        # Heading
        heading_label   =   ttk.Label(self.neuro_tran_tab,   text="Neuro-Style
Transfer", font=("Helvetica", 20, "bold"))
        heading_label.pack(pady=25)

        # Load the image
        image_path    =    "./Major    Project    -    Neuro    Style    and
Recognition/icons/nst.png"  # Replace with the actual path to your image
        image = Image.open(image_path)
        image = image.resize((275, 187))  # Adjust the size as necessary
        tk_image = ImageTk.PhotoImage(image)

        # Create a label for the image
        image_label = ttk.Label(self.neuro_tran_tab, image=tk_image)
        image_label.image = tk_image   # Keep a reference to avoid garbage
collection
        image_label.pack(pady=0)

        # Label frame with text
        label_frame    =    ttk.LabelFrame(self.neuro_tran_tab,    text="Style
Transfer")
        label_frame.pack(pady=10)

        info_label = ttk.Label(label_frame, text="Neural style transfer is an
optimization technique used to take two images, a content and a style reference
image, and blend them so the output image looks like the content image, but
“painted” in the style of the style reference image.\n    >> VGG-19\n    >>
Stylize \n   >> Process implementor \n   >> UI", wraplength=400)
        info_label.pack(padx=10, pady=10)

        # Configure font and size for the label frame text
        s = ttk.Style()
        s.configure('TLabelframe.Label', font=('Helvetica', 15, 'bold'))
        # Configure font and size for the content text
        s.configure('TLabel', font=('Helvetica', 11))
        button    =    ttk.Button(self.neuro_tran_tab,    text="Let's    Try!!",
command=self.run_neural_style_transfer)
        button.pack(side=tk.RIGHT, padx=(5, 10), pady=0)  # Place on the right
side
        button.place(relx=0.806, rely=0.88, anchor=tk.NE)  # Place on the corner
of the label frame
#NST Page Over

#Creating Gallery Page
    def create_gallery_widgets(self):
        # Heading
        heading_label     =     ttk.Label(self.gallery_tab,     text="Gallery",
font=("Helvetica", 20, "bold"))
        heading_label.pack(pady=25)

        # Create a Canvas widget for the gallery
        gallery_canvas = tk.Canvas(self.gallery_tab)
        gallery_canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Create a Scrollbar for the gallery
        scrollbar    =    ttk.Scrollbar(self.gallery_tab,    orient=tk.VERTICAL,
command=gallery_canvas.yview)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        # Configure the Canvas to use the Scrollbar
        gallery_canvas.configure(yscrollcommand=scrollbar.set)

        # Frame inside the Canvas to hold gallery items
        gallery_frame = tk.Frame(gallery_canvas)
        gallery_canvas.create_window((0,       0),       window=gallery_frame,
anchor=tk.NW)
        def show_image_popup(image_path):
            # Load the image
            image = Image.open(image_path)

            # Create a new window
            popup_window = tk.Toplevel()
            popup_window.title("Preview")

            # Resize image to fit within a maximum size (e.g., 800x800)
            max_size = (800, 800)
            image.thumbnail(max_size, Image.LANCZOS)

            # Display the image in a label
            photo = ImageTk.PhotoImage(image)
            label = tk.Label(popup_window, image=photo)
            label.image = photo  # Keep a reference to prevent image from being
garbage collected
            label.pack()

            # Function to close the popup window
            def close_popup():
                popup_window.destroy()
            # Button to close the popup window
            close_button     =     ttk.Button(popup_window,     text="Close",
command=close_popup)
            close_button.pack(pady=10)

        # Function to add gallery items (images)
        def add_gallery_item(path):
            # Check if the path is for an image
            if path.endswith(('.jpg', '.jpeg', '.png', '.gif')):
                # Load image
                image = Image.open(path)
                image.thumbnail((200, 200))  # Resize image
                photo = ImageTk.PhotoImage(image)
                # Create label to display image
                label = tk.Label(gallery_frame, image=photo)
                label.image = photo  # Keep a reference
                label.grid(row=add_gallery_item.row,
column=add_gallery_item.col, padx=10, pady=10)
                # Add click event to show image popup
                label.bind("<Button-1>",     lambda     event,     path=path:
show_image_popup(path))
                # Update row and column indices
                add_gallery_item.col += 1
                if add_gallery_item.col >= 3:
                    add_gallery_item.col = 0
                    add_gallery_item.row += 1
            else:
                print("Unsupported file format:", path)

        # Initialize row and column indices
        add_gallery_item.row = 0
        add_gallery_item.col = 0

        # Path to your assets folder
        assets_folder    =    "./Major    Project    -    Neuro    Style    and
Recognition/nst/results"

        # Gather paths of image files
        image_paths   =   [os.path.join(assets_folder,   file)   for   file   in
os.listdir(assets_folder) if file.endswith(('.jpg', '.jpeg', '.png', '.gif'))]

        # Add gallery items
        for path in image_paths:
            add_gallery_item(path)

        # Update the gallery size when the frame is changed
        def on_frame_configure(event):
            gallery_canvas.configure(scrollregion=gallery_canvas.bbox("all"))

        gallery_frame.bind("<Configure>", on_frame_configure)
#Gallery Page Over

#Creating About Us Page
    def create_about_us_widgets(self):
        # Heading
        heading_label = ttk.Label(self.about_us_tab, text="About Us", 
font=("Helvetica", 20, "bold"))
        heading_label.pack(pady=25)

        # Frame to hold images and names
        image_frame = tk.Frame(self.about_us_tab)
        image_frame.pack(pady=0)

        # Load and display the first image with name
        image1_path = "./Major Project - Neuro Style and 
Recognition/icons/1.png"  # Replace with the path to your first image
        image1 = Image.open(image1_path)
        image1.thumbnail((150, 150))  # Resize image
        photo1 = ImageTk.PhotoImage(image1)
        label_image1 = tk.Label(image_frame, image=photo1)
        label_image1.image = photo1
        label_image1.grid(row=0, column=0, padx=5, pady=5)

        label_name1 = ttk.Label(image_frame, text="Shaury Shobit")
        label_name1.grid(row=1, column=0, padx=10, pady=5)

        # Load and display the second image with name
        image2_path = "./Major Project - Neuro Style and 
Recognition/icons/2.png"  # Replace with the path to your second image
        image2 = Image.open(image2_path)
        image2.thumbnail((150, 150))  # Resize image
        photo2 = ImageTk.PhotoImage(image2)
        label_image2 = tk.Label(image_frame, image=photo2)
        label_image2.image = photo2
        label_image2.grid(row=0, column=1, padx=5, pady=5)

        label_name2 = ttk.Label(image_frame, text="Pragya Yadav")
        label_name2.grid(row=1, column=1, padx=10, pady=5)

        # Label frame with text
        label_frame = ttk.LabelFrame(self.about_us_tab, text="About Us")
        label_frame.pack(pady=0)
        info_label = ttk.Label(label_frame, text="We are both MCA final-year 
students from batch 2022-2024 at Vivekananda Institute of Professional 
Studies.\nOur project is based on the knowledge we have gained from our MCA. We 
used our past experience working on Python-related projects to implement this 
project.\nImplementing any application with a simple user interface is the main 
objective.\n  >>Learn new topics\n  >>Implement our modules\n  >>Integrate with 
UI\n  >>Easy and simple UI", wraplength=500, font=("Helvetica", 10))
        info_label.pack(padx=10, pady=10)
#About Us Page Over
def main():
    root = tk.Tk()
    app = NeuralApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Neuro Style Transfer with GUI



Content Image

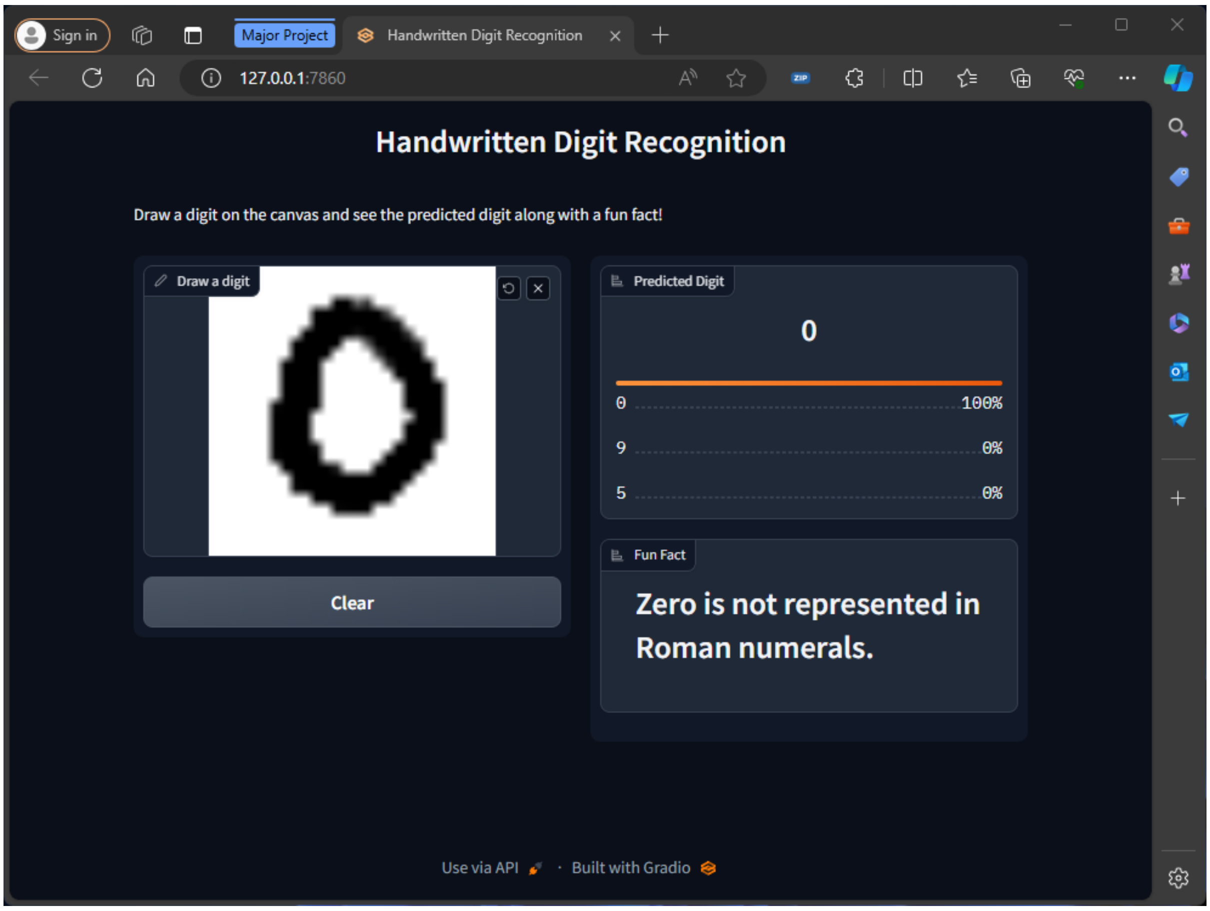
Style Image

```
import os
import numpy as np
from PIL import Image, ImageTk
from argparse import ArgumentParser
from collections import OrderedDict
from stylize import stylize  # Assuming this is where the stylize function is
defined
import datetime  # Importing the datetime module
import tkinter as tk
from tkinter import filedialog

CONTENT_WEIGHT = 5e0
STYLE_WEIGHT = 5e2
TV_WEIGHT = 1e2
STYLE_LAYER_WEIGHT_EXP = 1
LEARNING_RATE = 1e1
BETA1 = 0.9
BETA2 = 0.999
```

```
EPSILON = 1e-08
VGG_PATH = "./Major Project - Neuro Style and Recognition/nst/imagenet-vgg-
verydeep-19.mat"
POOLING = "max"

def build_parser():
    parser = ArgumentParser()
    parser.add_argument("--content", required=True, help="Content image")
    parser.add_argument("--styles", nargs="+", required=True, help="Style
images")
    # Removed output argument
    parser.add_argument("--iterations", type=int, default=1000, help="Number
of iterations")
    parser.add_argument("--content-weight", type=float,
default=CONTENT_WEIGHT, help="Content weight")
    parser.add_argument("--style-weight", type=float, default=STYLE_WEIGHT,
help="Style weight")
    parser.add_argument("--style-layer-weight-exp", type=float,
default=STYLE_LAYER_WEIGHT_EXP, help="Style layer weight exponent")
    parser.add_argument("--tv-weight", type=float, default=TV_WEIGHT,
help="Total variation regularization weight")
    parser.add_argument("--learning-rate", type=float, default=LEARNING_RATE,
help="Learning rate")
    parser.add_argument("--beta1", type=float, default=BETA1, help="Adam:
beta1 parameter")
    parser.add_argument("--beta2", type=float, default=BETA2, help="Adam:
beta2 parameter")
    parser.add_argument("--epsilon", type=float, default=EPSILON, help="Adam:
epsilon parameter")
    parser.add_argument("--network", default=VGG_PATH, help="Path to network
parameters")
    parser.add_argument("--pooling", default=POOLING, help="Pooling layer
configuration")
    parser.add_argument("--overwrite", action="store_true", help="Write file
even if it exists")
    return parser

def run_neural_style_transfer(content_path, style_paths, iterations=1000,
content_weight=CONTENT_WEIGHT,
                              style_weight=STYLE_WEIGHT,
style_layer_weight_exp=STYLE_LAYER_WEIGHT_EXP,
                              tv_weight=TV_WEIGHT,
learning_rate=LEARNING_RATE, beta1=BETA1, beta2=BETA2,
                              epsilon=EPSILON, network=VGG_PATH,
pooling=POOLING):
    os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
    content_image = imread(content_path)
    style_images = [imread(style) for style in style_paths]

    style_blend_weights = [1.0 / len(style_images) for _ in style_images]  #
Equal weights for simplicity

    # Removed initial handling

    initial = None
    initial_noiseblend = 0.0  # Setting initial noise blend to 0.0 by default
    preserve_colors = False
    content_weight_blend = 1

    loss_arrs = None
    for iteration, image, loss_vals in stylize(
        network=network,
        initial=initial,
        initial_noiseblend=initial_noiseblend,
        content=content_image,
        styles=style_images,
        preserve_colors=preserve_colors,
        content_weight_blend=content_weight_blend,
        style_blend_weights=style_blend_weights,
        iterations=iterations,
        content_weight=content_weight,
        style_weight=style_weight,
        style_layer_weight_exp=style_layer_weight_exp,
        tv_weight=tv_weight,
        learning_rate=learning_rate,
        beta1=beta1,
        beta2=beta2,
        epsilon=epsilon,
        pooling=pooling,
    ):
        pass

    # Get the current date and time
    current_datetime = datetime.datetime.now()
    # Construct the output file name with date and time
    output_file = "./Major Project - Neuro Style and
Recognition/nst/results/Neuro-Style_" +
current_datetime.strftime("%Y-%m-%d_%H-%M-%S") + ".jpg"
    # Save the image
    imsave(output_file, image)

def imread(path):
    img = np.array(Image.open(path)).astype(np.float32)
    return img

def imsave(path, img):
    img = np.clip(img, 0, 255).astype(np.uint8)
    Image.fromarray(img).save(path, quality=95)

def select_content_image():
    path = filedialog.askopenfilename()
    if path:
        content_entry.delete(0, tk.END)
        content_entry.insert(0, path)
        display_image(content_image_frame, path)

def select_style_images():
    paths = filedialog.askopenfilenames()
    if paths:
        style_entry.delete(0, tk.END)
        style_entry.insert(0, ", ".join(paths))
        for path in paths:
            display_image(style_image_frame, path)

def display_image(frame, path):
    img = Image.open(path)
    img.thumbnail((200, 200))
    img = ImageTk.PhotoImage(img)
    panel = tk.Label(frame, image=img)
    panel.image = img
    panel.pack()

def start_neural_style_transfer():
    content_path = content_entry.get()
    style_paths = style_entry.get().split(", ")
    run_neural_style_transfer(content_path, style_paths)

# Create Tkinter GUI
root = tk.Tk()
root.title("Neural Style Transfer")

content_frame = tk.Frame(root)
content_frame.pack(fill=tk.BOTH, expand=True)

content_label = tk.Label(content_frame, text="Content Image:")
content_label.pack(side=tk.LEFT)
content_entry = tk.Entry(content_frame)
content_entry.pack(side=tk.LEFT, padx=5)
```

```
content_button = tk.Button(content_frame, text="Browse",
command=select_content_image)
content_button.pack(side=tk.LEFT)

style_frame = tk.Frame(root)
style_frame.pack(fill=tk.BOTH, expand=True)

style_label = tk.Label(style_frame, text="Style Images:")
style_label.pack(side=tk.LEFT)
style_entry = tk.Entry(style_frame)
style_entry.pack(side=tk.LEFT, padx=5)
style_button = tk.Button(style_frame, text="Browse",
command=select_style_images)
style_button.pack(side=tk.LEFT)

start_button = tk.Button(root, text="Start Neural Style Transfer",
command=start_neural_style_transfer)
start_button.pack()

# Frames to display selected images
content_image_frame = tk.Frame(root)
content_image_frame.pack(fill=tk.BOTH, expand=True)

style_image_frame = tk.Frame(root)
style_image_frame.pack(fill=tk.BOTH, expand=True)

root.mainloop()
```

Digit Recognition



Handwritten Digit Recognition

Draw a digit on the canvas and see the predicted digit along with a fun fact!

```
0
0 ........ 100%
9 ........ 0%
5 ........ 0%
```

Zero is not represented in Roman numerals.

Model Training

```
import tensorflow as tf
from tensorflow.keras import layers, models

(train_images, train_labels), (test_images, test_labels) =
tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') /
255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

train_labels = tf.keras.utils.to_categorical(train_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)

model = models.Sequential()
model.add(layers.Conv2D(64, (3,3), activation='relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation='relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))



model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=10, batch_size=64,
validation_data=(test_images, test_labels))

model.save('digit.h5')
```

GUI for model

```
import gradio as gr
import tensorflow as tf
import os
import numpy as np
import random

# Load the trained model
model = tf.keras.models.load_model('./Major Project - Neuro Style and
Recognition/digit/digit.h5')

# Function to fetch random fun fact based on predicted digit
def get_fun_fact(predicted_digit):
    file_path = f'./Major Project - Neuro Style and
Recognition/digit/fun_facts/{predicted_digit}.txt'
    if os.path.exists(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            facts = file.readlines()
            return random.choice(facts).strip()
    else:
        return "No fun facts available for this digit."

# Function to recognize digit
def recognize_digit(image):
    if image is not None:
        image = image.reshape(1, 28, 28, 1).astype('float32') / 255
        prediction = model.predict(image)[0]
        top_predicted_digit = str(np.argmax(prediction))
        top_3_indices = np.argsort(prediction)[::-1][:3]
        top_3_probs = [float(prediction[i]) for i in top_3_indices]  # Convert
NumPy float32 to Python float
```

```
        top_3_digits = [str(i) for i in top_3_indices]
        fun_fact = get_fun_fact(top_predicted_digit)
        return {digit: prob for digit, prob in zip(top_3_digits,
top_3_probs)}, fun_fact
    else:
        return {}, ''

# Create a Gradio interface
iface = gr.Interface(
    fn=recognize_digit,
    inputs=gr.inputs.Image(shape=(28, 28), image_mode='L', invert_colors=True,
source='canvas', label="Draw a digit"),
    outputs=[
        gr.outputs.Label(num_top_classes=3, label="Predicted Digit"),
        gr.outputs.Label(label="Fun Fact")
    ],
    theme="compact",
    title="Handwritten Digit Recognition",
    description="Draw a digit on the canvas and see the predicted digit along
with a fun fact!",
    allow_flagging=False,
    live=True
)

iface.launch(share=True)
```

Limitations of the System

Existing Systems:

- Designed for high-end systems, requiring substantial computational resources.
- Typically lack graphical user interfaces (GUIs), making interaction and usability challenging.
- May include numerous features, many of which are not fully utilized by average users.
- Resource-intensive operations often result in longer processing times, especially on average systems.
- Primarily targeted towards researchers or server-based applications, with limited accessibility for general users.
- Usability may be hindered by complex configurations and technical requirements.

Limitations overcome by this new “Neuro Style & Recognition” project

Neuro System:

- Tailored for average systems, ensuring compatibility and efficient resource utilization.
- Incorporates a user-friendly graphical interface (UI), enhancing accessibility and ease of interaction.
- Focuses on including only essential features, optimizing usability and streamlining functionality.
- Prioritizes time efficiency, enabling swift execution of tasks even on standard hardware configurations.
- Designed to be user-centric, accessible to individuals with varying technical backgrounds and requirements.
- Offers straightforward operation and intuitive controls, making it suitable for a wide range of users and applications.

Conclusion of the System

In conclusion, the Neuro-Style Transfer and Digit Recognition system represents a significant advancement in the field of machine learning applications, particularly in the domains of artistic style transfer and digit recognition. By combining sophisticated neural network algorithms with user-friendly interfaces, the system offers a powerful yet accessible tool for both creative expression and educational purposes.

The Neuro-Style Transfer component allows users to seamlessly blend content images with various artistic styles, enabling the creation of visually stunning artworks with minimal effort. The inclusion of a graphical user interface (GUI) enhances usability, making the style transfer process intuitive and engaging for artists, designers, and hobbyists alike. Moreover, the system's focus on efficiency ensures that even users with average systems can enjoy fast and responsive performance.

Similarly, the Digit Recognition component provides a fun and interactive way for young children to learn and practice recognizing digits. With its intuitive drawing canvas and real-time feedback, the system offers an engaging learning experience that complements traditional educational methods. By incorporating educational content and fun facts, the system enhances the learning process and encourages continuous improvement.

Future scope of the System

Looking ahead, there are several avenues for further development and enhancement of the system:

- **Advanced Style Transfer Techniques:** Explore and integrate state-of-the-art style transfer algorithms to further improve the quality and diversity of stylized images.
- **Expand Digit Recognition Capabilities:** Extend the digit recognition system to recognize other types of symbols, such as letters, shapes, or even handwritten words.
- **Integration with External Datasets:** Incorporate additional datasets for training the digit recognition model, allowing for improved accuracy and robustness across different writing styles and languages.
- **Customization and Personalization:** Implement features that allow users to customize and personalize their style transfer and digit recognition experiences, such as adjusting parameters or saving preferences.
- **Mobile Compatibility:** Adapt the system for mobile platforms, enabling users to access and use the applications on smartphones and tablets for greater convenience and flexibility.
- **Community Contributions and Collaboration:** Foster a community-driven approach to development by encouraging contributions from users, developers, and researchers, ensuring ongoing innovation and improvement.

By pursuing these avenues for future development, the Neuro-Style Transfer and Digit Recognition system can continue to evolve and meet the needs of its users, while pushing the boundaries of creativity and education in the realm of machine learning applications.

SCREENSHOTS

These are the images and screenshots

for "Neuro - Style and Recognition"

Images of Content and Styles

Screenshots of:

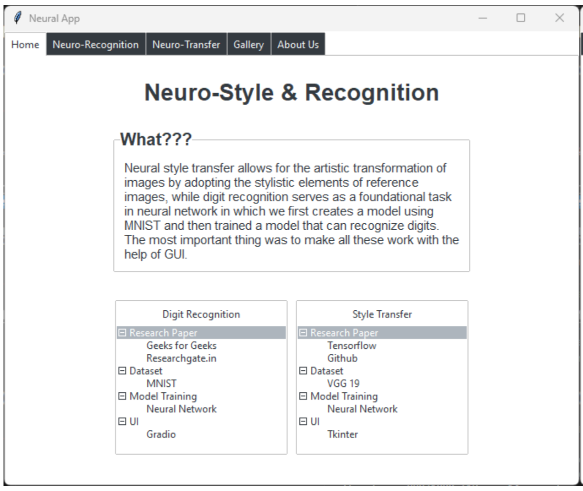
>> Admin Page

>> Neuro Style

>> Neuro Recognition

In the end check these outputs of the

project



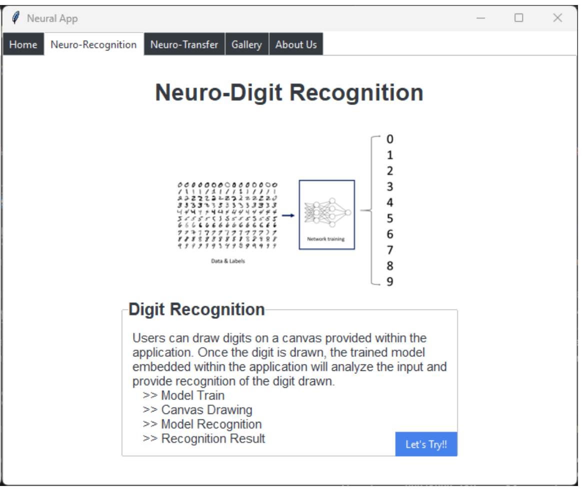
Neuro-Style & Recognition

What???

Neural style transfer allows for the artistic transformation of images by adopting the stylistic elements of reference images, while digit recognition serves as a foundational task in neural network in which we first creates a model using MNIST and then trained a model that can recognize digits. The most important thing was to make all these work with the help of GUI.

Digit Recognition
- Research Paper
  - Geeks for Geeks
  - Researchgate.in
- Dataset
  - MNIST
- Model Training
  - Neural Network
- UI
  - Gradio

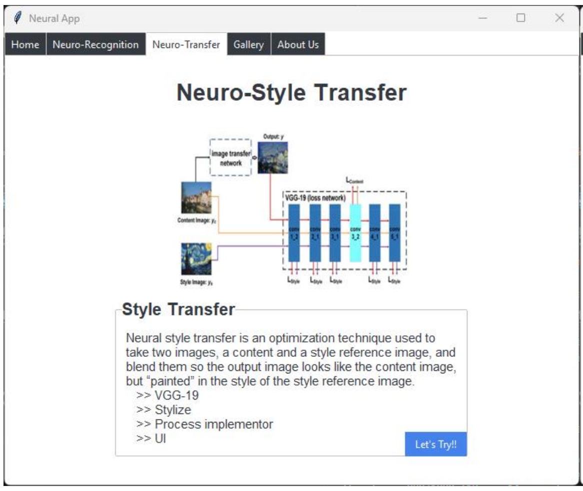
Style Transfer
- Research Paper
  - Tensorflow
  - Github
- Dataset
  - VGG 19
- Model Training
  - Neural Network
- UI
  - Tkinter



Neuro-Digit Recognition

Digit Recognition

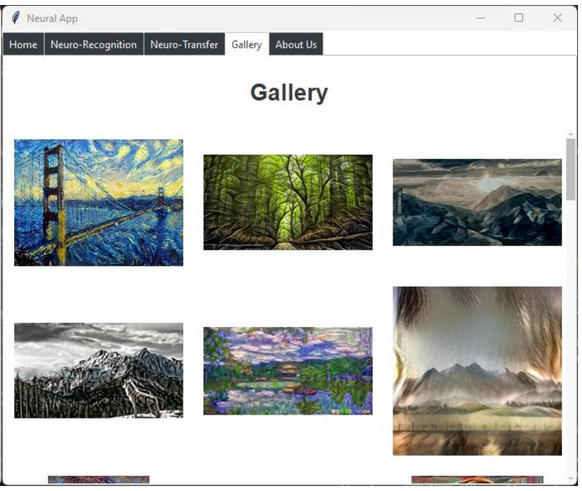
Users can draw digits on a canvas provided within the application. Once the digit is drawn, the trained model embedded within the application will analyze the input and provide recognition of the digit drawn.
>> Model Train
>> Canvas Drawing
>> Model Recognition
>> Recognition Result

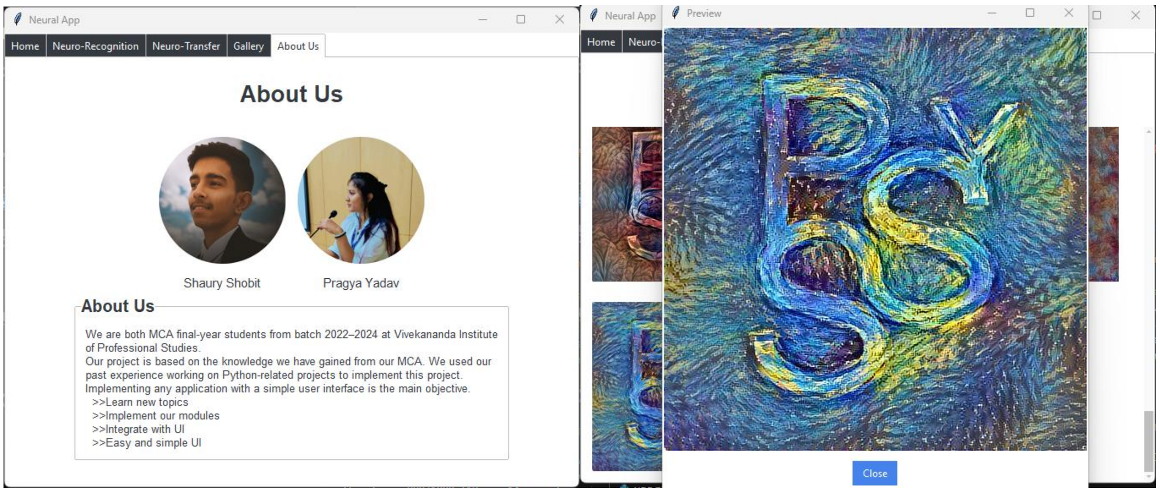


Neuro-Style Transfer

Style Transfer

Neural style transfer is an optimization technique used to take two images, a content and a style reference image, and blend them so the output image looks like the content image, but "painted" in the style of the style reference image.
>> VGG-19
>> Stylize
>> Process implementor
>> UI

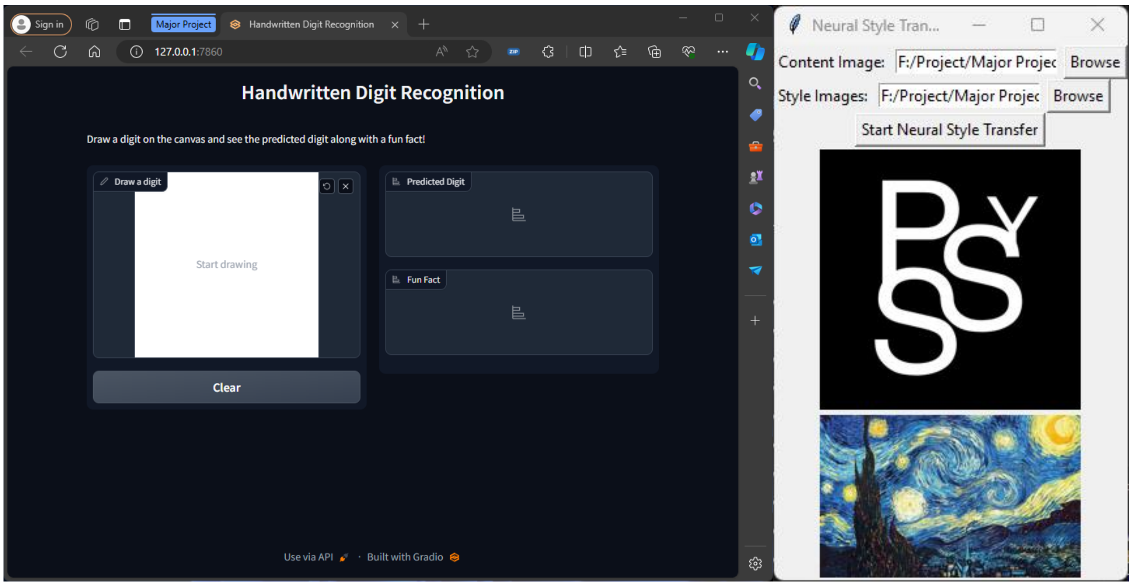




About Us

About Us

We are both MCA final-year students from batch 2022–2024 at Vivekananda Institute of Professional Studies.
Our project is based on the knowledge we have gained from our MCA. We used our past experience working on Python-related projects to implement this project.
Implementing any application with a simple user interface is the main objective.
>>Learn new topics
>>Implement our modules
>>Integrate with UI
>>Easy and simple UI

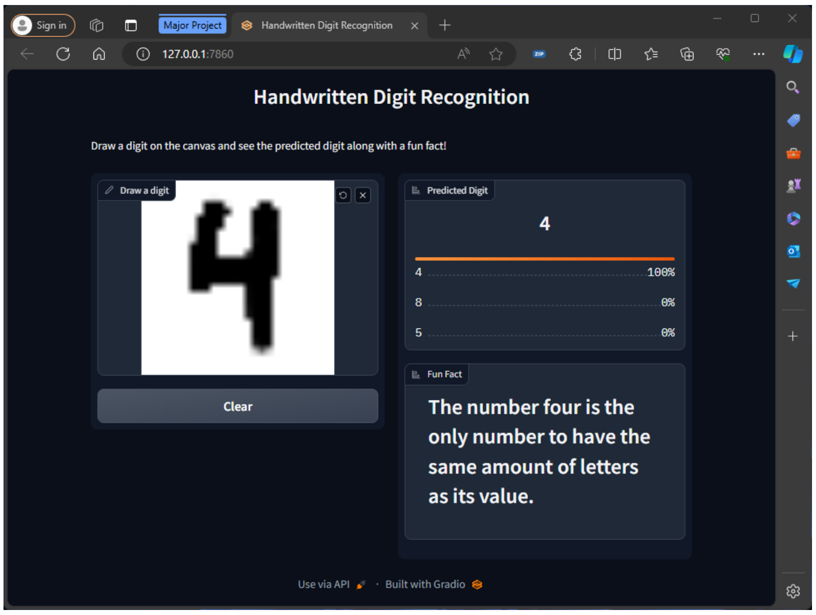


Handwritten Digit Recognition

Draw a digit on the canvas and see the predicted digit along with a fun fact!

Content Image: F:/Project/Major Projec

Style Images: F:/Project/Major Projec



Handwritten Digit Recognition

Draw a digit on the canvas and see the predicted digit along with a fun fact!

4

```
4 ............................ 100%
8 ..............................  0%
5 ..............................  0%
```

The number four is the only number to have the same amount of letters as its value.

Bibliography

"A Neural Algorithm of Artistic Style" by Leon Gatys et al., originally released to ArXiv 2015

“Understanding VGG19” - by various websites (researchgate.net ; arxiv.org ; iq.opengenus.org)

Github - anishathalye/neural-style

Tensorflow - https://www.tensorflow.org/tutorials/generative/style_transfer

https://www.tensorflow.org/datasets